

```

B[0] = 0;
T = Rpic / NUM_SLICE × 0.5;
for(slice = 1; slice < NUM_SLICE; slice++) {
    /* Coding one slice */
    B[slice] = B[slice - 1] + G[slice] - R[slice];
    if( B[slice] > T && B[slice] > B[slice - 1] && Qs < 31) Qs += 1;
    else if(B[slice] < -T && B[slice] < B[slice - 1] && Qs > 1) Qs -= 1;
}

```

In addition, if the number of generated bits in an I-picture exceeds 40% of  $C_{GOP}$ , or the content of the output data buffer exceeds 98% of its size, all coefficients will be set to 0 so that no codes will be generated.

#### 2.10.4.2 Rate control in P0-picture

A hypothetical buffer is also used in coding a P0-picture. However, its purpose is to limit the maximum number of generated bits, so that the sum of generated bits in I and P0-pictures does not exceed 50% of  $C_{GOP}$ . Therefore, the number of taken bits from the buffer at each slice is set to be one NUM\_SLICE-th of the usable number of bits for the P0-picture. If the buffer content exceeds a threshold T, which is equal to the number of taken bits from the buffer, the quantizer scale is increased by one. These are summarized as follows:

```

B[0] = 0;
T = (CGOP × 0.49 - GI) / NUM_SLICE;
for(slice = 1; slice < NUM_SLICE; slice++) {
    /* Coding one slice */
    B[slice] = B[slice - 1] + G[slice] - T;
    if(B[slice] > T && Qs < 31) Qs += 1;
}

```

In addition, if the sum of generated bits in I and P0-pictures exceeds 49% of  $C_{GOP}$ , or the content of the output data buffer exceed 98% of its size, then the mode of all Macroblocks will be set to the "skipped" mode so that no codes will be generated.

#### 2.10.4.3 Rate control in B-picture

The same control rule as that in P1 and P2-pictures is applied except for the following two points.

- 1) The quantizer scale is updated every 10 slices.
- 2) The quantizer scale is updated whenever the buffer content is larger (smaller) than the threshold T (-T) even if the buffer content is smaller (larger) than that at the previous slice.

These are summarized as follows:

```

B[0] = 0;
T = Rpic / NUM_SLICE × 1.5;
for( slice = 1; slice < NUM_SLICE; slice++) {
    /* Coding one slice */
    B[slice] = B[slice - 1] + G[slice] - R[slice];
    if(slice % 10 == 0)
        if(B[slice] > T && Qs < 31) Qs += 1;
        else if(B[slice] < -T && Qs > 1) Qs -= 1;
}

```

In addition, if the content of the output data buffer exceeds 98% of its size, then the mode of all Macroblocks will be set to the "skipped" mode so that no codes will be generated.

### 2.10.5 Rate control after scene change

The number of generated bits in a P-picture after scene change is much larger than those in other P-pictures. Therefore, the following rules are exceptionally applied to rate control when a scene change occurs.

It is determined that a scene change has occurred if more than 50 % of the Macroblocks in the first four slices in a P1-picture are coded by the Intra mode. The quantizer scale in a P1-picture where a scene change has been detected is limited to the quantizer scale at the first slice in the P1-picture + 3, and the quantizer scale in the next P2-picture is fixed to its initial value defined in Table 2-5. The numbers of allocated bits for remaining P1, P2, and B-pictures within a GOP are determined as follows, so that the part of bits for the next GOP can be used for them:

$$R_{P1} = ( R_{GOP} - G_C + C_{GOP} - G_I' \times A_{P1\_scene\_change} / A_I' - G_{P0}' \times A_{P2\_scene\_change} / A_{P0}' ) / ( (N_{P1} + 3) + K_{P1\_P2} \times (N_{P2} + 3) + K_{P1\_B} \times (N_B + 16) )$$

$$R_{P2} = K_{P1\_P2} \times R_{P1}$$

$$R_B = K_{P1\_B} \times R_{P1}$$

where  $G_I'$  and  $G_{P0}'$  are the number of generated bits in the most recently coded I and P0-pictures, respectively.

### 3. Specification for coded bit stream

#### 3.1 Bit stream syntax specification

##### 3.1.1 Start codes

name	hexadecimal value
picture_start_code	00000100
slice_start_codes (including slice_vertical_positions)	00000101
	through
reserved	000001AF
reserved	000001B0
user_data_start_code	000001B1
sequence_header_code	000001B2
sequence_error_code	000001B3
extension_start_code	000001B4
reserved	000001B5
sequence_end_code	000001B6
group_start_code	000001B7
system_start_codes	000001B8
	000001B9
	through
	000001FF

##### 3.1.2 Definition of next\_start\_code function

```

next_start_code() {
    while (!bytealigned())
        zero_bit
    while (nextbits() != '0000 0000 0000 0000
                                0000 0001')
        zero_byte
}

```

1            "0"  
8            "00000000"

##### 3.1.3 Video sequence layer

```

video_sequence() {
    next_start_code()
    do {
        sequence_header()
        do {
            group_of_pictures()
        } while (nextbits() == group_start_code)
    } while (nextbits() == sequence_header_code)
    sequence_end_code
}

```

32            bslbf

### 3.1.4 Sequence header

```

sequence_header() {
    sequence_header_code           32      bslbf
    horizontal_size                 12      uimsbf
    vertical_size                   12      uimsbf
    pel_aspect_ratio                4       uimsbf
    picture_rate                    4       uimsbf
    bit_rate                        18      uimsbf
    marker_bit                      1       "1"
    vbv_buffer_size                 10      uimsbf
    constrained_parameter_flag      1
    load_intra_quantizer_matrix     1
    if ( load_intra_quantizer_matrix )
        intra_quantizer_matrix[64]      8*64    uimsbf
    load_non_intra_quantizer_matrix 1
    if ( load_non_intra_quantizer_matrix )
        non_intra_quantizer_matrix[64]  8*64    uimsbf
    next_start_code()
    if (nextbits() == extension_start_code) {
        extension_start_code           32      bslbf
        while ( nextbits () != '0000 0000 0000 0000 0000
                                0001' ) {
            sequence_extension_data      8
        }
        next_start_code()
    }
    if (nextbits() == user_data_start_code) {
        user_data_start_code            32      bslbf
        while ( nextbits() != '0000 0000 0000 0000 0000
                                0001' ) {
            user_data                    8
        }
        next_start_code()
    }
}
}

```

### 3.1.5 Group of pictures layer

```

group_of_pictures() {
    group_start_code                32 bits    bslbf
    time_code                        25
    closed_gop                       1
    broken_link                      1
    next_start_code()
}

```

```

    if ( nextbits() == extension_start_code ) {
        extension_start_code          32          bslbf
        while ( nextbits() != '0000 0000 0000 0000
                                0000 0001' ) {
            group_extension_data      8
        }
        next_start_code()
    }
    if ( nextbits() == user_data_start_code ) {
        user_data_start_code          32          bslbf
        while ( nextbits() != '0000 0000 0000 0000
                                0000 0001' ) {
            user_data                  8
        }
        next_start_code()
    }
    do {
        picture()
    } while ( nextbits() == picture_start_code )
}

```

### 3.1.6 Picture layer

```

picture() {
    picture_start_code
    temporal_reference          32 bits          bslbf
    picture_coding_type         10              uimsbf
    vbv_delay                   3               uimsbf
    if ( picture_coding_type >= 5 ||
        picture_coding_type == 3 ) {
        full_pel_forward_vector  16             uimsbf
        forward_f                1
    }
    if ( picture_coding_type == 3 ||
        picture_coding_type == 7 ) {
        full_pel_backward_vector  1
        backward_f                3
    }
    while ( nextbits() == '1' ) {
        extra_bit_picture         1
        extra_information_picture  8              "1"
    }
    extra_bit_picture            1
    next_start_code()             1              "0"

    if ( nextbits() == extension_start_code ) {
        extension_start_code      32          bslbf
        while ( nextbits() != '0000 0000 0000 0000

```

```

                                0000 0001' ) {
picture_extension_data      8
    }
next_start_code()
}
if ( nextbits() == user_data_start_code ) {
    user_data_start_code      32      bslbf
    while ( nextbits() != '0000 0000 0000 0000
                                0000 0001' ) {
        user_data      8
    }
    next_start_code()
}
do {
    slice()
} while ( nextbits() == slice_start_code )
}

```

### 3.1.7 Slice layer

```

slice() {
    slice_start_code      32 bit      bslbf
    quantizer_scale      5      uimsbf
    while ( nextbits() == '1' ) {
        extra_bit_slice      1      "1"
        extra_information_slice      8
    }
    extra_bit_slice      1      "0"
    do {
        macroblock()
    } while ( nextbits() != '000 0000 0000 0000
                                0000 0000' )
    next_start_code()
}

```

### 3.1.8 Macroblock layer

```

macroblock() {
    while ( nextbits() == '0000 0001 111' )
        macroblock_stuffing      11 bits      vlclbf
    while ( nextbits() == '0000 0001 000' )
        macroblock_escape      11      vlclbf
        macroblock_address_increment      1-11      vlclbf
        macroblock_type      1-6      vlclbf
    if ( macroblock_quant )
        quantizer_scale      5      uimsbf
    if ( macroblock_motion_forward ) {

```

if ( picture_coding_type != 5 )		
field_or_frame_forward	1	
if ( picture_coding_type == 6		
picture_coding_type == 7		
( picture_coding_type == 3 &&		
field_or_frame_forward == 0 ) )		
select_mv_forward	1	
motion_horizontal_forward	1-14	vlc1bf
motion_vertical_forward	1-14	vlc1bf
if ( field_or_frame_forward == 1 ) {		
dmv_horizontal_forward	1-2	vlc1bf
dmv_vertical_forward	1-2	vlc1bf
}		
}		
if ( macroblock_motion_backward ) {		
if ( picture_coding_type == 3 )	1	
field_or_frame_backward		
if ( picture_coding_type == 3 &&		
field_or_frame_backward == 0 )		
select_mv_backward	1	
motion_horizontal_backward	1-14	vlc1bf
motion_vertical_backward	1-14	vlc1bf
if ( field_or_frame_backward == 1 ) {		
dmv_horizontal_backward	1-2	vlc1bf
dmv_vertical_backward	1-2	vlc1bf
}		
}		
if ( macroblock_pattern )		
coded_block_pattern	1-8	vlc1bf
for ( i=0; i<4; i++ )		
block( i )		
if ( picture_coding_type == 4 )		
end_of_macroblock	1	"1"
}		

### 3.1.9 Block layer

block( i ) {		
if ( pattern_code[ i ] ) {		
if ( macroblock_intra ) {		
if ( i < 2 ) {		
dct_dc_size_luminance	2-7	vlc1bf
if ( dct_dc_size_luminance != 0 )		
dct_dc_differential	1-8	uimsbf
}		
else {		
dct_dc_size_chrominance	2-8	vlc1bf
if ( dct_dc_size_chrominance != 0 )		
}		
}		

dct_dc_differential	1-8	uimsbf
}		
}		
else {		
dct_coeff_first	2-28	vlc1bf
}		
if (picture_coding_type != 4) {		
while (nextbits() != '010')		
dct_coeff_next	3-28	vlc1bf
end_of_block	3	"010"
}		
}		
}		

### 3.2 Semantic meaning and use of retrieved data elements

#### 3.2.1 Video sequence layer

The meaning of the following data element is the same as in MPEG1 C.D. on 5/31/91.

`sequence_end_code`

#### 3.2.2 Sequence header

`load_intra_quantizer_matrix` -- This is a one-bit flag which is set to "1" if `intra_quantizer_matrix` follows. If it is set to "0" then the default values defined below are used until the next occurrence of the sequence header.

0	8	11	14	18	19	21	26
8	8	14	16	19	21	26	29
11	14	18	19	21	26	26	30
14	14	18	19	21	26	29	32
14	18	19	21	24	27	32	40
18	19	21	24	27	32	40	50
18	19	21	26	30	38	48	61
19	21	27	30	38	48	61	75

`intra_quantizer_matrix` -- This is a list of 64 8-bit unsigned integers. The new values, stored in the zigzag scanning order same as that in MPEG1 C.D., replace the default values. The value for `intra_quant[0][0]` shall always be 0. The new values shall be in effect until the next occurrence of a sequence header.

`load_non_intra_quantizer_matrix` -- This is a one-bit flag which is set to "1" if `non_intra_quantizer_matrix` follows. If it is set to "0" then the default values defined below are used until the next occurrence of the sequence header.

8	10	12	14	16	18	20	22
9	11	13	15	17	19	21	23
10	12	14	16	18	20	22	24
11	13	15	17	19	21	23	25
12	14	16	18	20	22	24	26
13	15	17	19	21	23	25	27
14	16	18	20	22	24	26	28
15	17	19	21	23	25	27	29

`non_intra_quantizer_matrix` -- This is a list of 64 8-bit unsigned integers. The new values, stored in the zigzag scanning order same as that in MPEG1 C.D., replace the default values. The new values shall be in effect until the next occurrence of a sequence header.

The meaning of the following data elements is the same as the MPEG1 C.D. on 5/31/91.

<b>sequence_header_code</b>	<b>horizontal_size</b>
<b>vertical_size</b>	<b>pel_aspect_ratio</b>
<b>picture_rate</b>	<b>bit_rate</b>
<b>marker_bit</b>	<b>vbv_buffer_size</b>
<b>constrained_parameters_flag</b>	<b>extension_start_code</b>
<b>sequence_extension_data</b>	<b>user_data_start_code</b>
<b>user_data</b>	

In our simulation, the data elements mentioned above are set as follows :

**horizontal\_size** -- 720.  
**vertical\_size** -- 240.  
**pel\_aspect\_ratio** -- '1100', for CCIR601, 525 lines.  
**picture\_rate** -- '1000', for 60 pictures/second.  
**bit\_rate** -- either 10000 ( for 4 Mbits/sec. ) or 22500 ( for 9 Mbits/sec. ).  
**vbv\_buffer\_size** -- unused.  
**constrained\_parameters\_flag** -- unused.

### 3.2.3 Group of pictures layer

The meaning of the following data elements is the same as in MPEG1 C.D. on 5/31/91.

<b>group_start_code</b>	<b>time_code</b>
<b>closed_gop</b>	<b>broken_link</b>
<b>extension_start_code</b>	<b>group_extension_data</b>
<b>user_data_start_code</b>	<b>user_data</b>

### 3.2.4 Picture layer

**picture\_coding\_type** -- The **picture\_coding\_type** identifies whether a picture is an intra-coded picture(I), predictive-coded picture(P0, P1, or P2), bidirectionally predictive-coded picture(B), or intra-coded with only dc coefficients (D) according to the following table. D-pictures shall never be included in the same video sequence as other picture coding types.

picture_coding_type	coding method
000	forbidden
001	intra-coded (I)
010	reserved
011	bidirectionally-predictive (B)
100	dc intra-coded (D)
101	predictive-coded (P0)
110	predictive-coded (P1)
111	predictive-coded (P2)

The meaning of the following data elements is the same as in MPEG1 C.D. on 5/31/91.

<b>picture_start_code</b>	<b>temporal_reference</b>
<b>vbv_delay</b>	<b>full_pel_forward_vector</b>
<b>full_pel_backward_vector</b>	<b>extra_bit_picture</b>
<b>extra_information_picture</b>	<b>extension_start_code</b>
<b>picture_extension_data</b>	<b>user_data_start_code</b>
<b>user_data</b>	

The meaning of the following data elements is the same as in MPEG1 C.D. on 12/18/90.

<b>forward_f</b>	<b>backward_f</b>
------------------	-------------------

In our simulation, the data elements mentioned above are set as follows :

**vbv\_delay** -- unused.  
**full\_pel\_forward\_vector** -- 0.  
**full\_pel\_backward\_vector** -- 0.

### 3.2.5 Slice layer

**quantizer\_scale** -- An unsigned integer in the range 1 to 31 is used to scale the reconstruction level of the retrieved DCT coefficient levels. The decoder shall use this value in the slice if another **quantizer\_scale** is not encountered at the macroblock layer.

The meaning of the following data elements is the same as in MPEG1 C.D. on 5/31/91.

<b>slice_start_code</b>	<b>slice_vertical_position</b>
<b>extra_bit_slice</b>	<b>extra_information_slice</b>

### 3.2.6 Macroblock layer

**quantizer\_scale** -- An unsigned integer in the range 1 to 31 is used to scale the reconstruction level of the retrieved DCT coefficient levels. The decoder shall use this value only one macroblock. The presence of a **quantizer\_scale** is determined from the **macroblock\_type**.

**field\_or\_frame\_forward** -- This is a one-bit flag for selecting the interpolation mode for a forward reference picture. This flag is set to 0 if the picture being decoded shall be reconstructed by referencing the picture interpolated by the field mode. This flag is set to 1 if the picture being decoded shall be reconstructed by referencing a picture interpolated by the frame mode. If the **picture\_coding\_type** is a P0-picture, this flag is absent and shall be set to 0.

**select\_mv\_forward** -- This is a one-bit flag for selecting a forward reference picture. This flag is set to 0 if the reference picture reconstructed by forward motion vector is an even field picture. This flag is set to 1 if the reference picture reconstructed by the forward motion vector is an odd field picture. If the **picture\_coding\_type** is a P0-picture, this flag is absent and shall be set to 0. If the **picture\_coding\_type** is a B-picture and the

**field\_or\_frame\_forward** is 1, this flag is absent and shall be set so that the picture selected by this flag has the same interlacing phase.

**motion\_horizontal\_forward** -- Forward horizontal motion vector for a selected field picture coded in half pel units according to Table 7-4 in Section 7.

**motion\_vertical\_forward** -- Forward vertical motion vector for a selected field picture coded in half pel units according to Table 7-4 in Section 7.

**dmv\_horizontal\_forward** -- Difference between a horizontal motion vector reconstructed from a forward horizontal motion vector for the selected field picture and a forward horizontal motion vector for the not selected field picture coded in one pel units according to Table 7-4g in Section 7. This element is present only if the **field\_or\_frame\_forward** flag is set to 1.

**dmv\_vertical\_forward** -- Difference between a vertical motion vector reconstructed from a forward vertical motion vector for the selected field picture and a forward vertical motion vector for the not selected field picture coded in one pel units according to Table 7-4g in Section 7. This element is present only if the **field\_or\_frame\_forward** flag is set to 1.

**field\_or\_frame\_backward** -- This is a one-bit flag for selecting the interpolation mode for a backward reference picture. This flag is set to 0 if the picture being decoded shall be reconstructed by referencing a picture interpolated by the field mode. This flag is set to 1 if the picture being decoded shall be reconstructed by referencing a picture interpolated by the frame mode. If the **picture\_coding\_type** is a P2-picture, this flag is absent and shall be set to 0.

**select\_mv\_backward** -- This is a one-bit flag for selecting a backward reference picture. This flag is set to 0 if a reference picture reconstructed by a backward motion vector is an even field picture. This flag is set to 1 if the reference picture reconstructed by a backward motion vector is an odd field picture. If the **picture\_coding\_type** is a P2-picture, this flag is absent and shall be set to 0. If the **picture\_coding\_type** is a B-picture and **field\_or\_frame\_backward** is 1, this flag is absent and shall be set so that the picture selected by this flag has the same interlacing phase.

**motion\_horizontal\_backward** -- Backward horizontal motion vector for a selected field picture coded in half pel units according to Table 7-4 Section 7.

**motion\_vertical\_backward** -- Backward vertical motion vector for a selected field picture coded in half pel units according to Table 7-4 in Section 7.

**dmv\_horizontal\_backward** -- Difference between a horizontal motion vector reconstructed from a backward vertical motion vector for the selected field picture and a backward horizontal motion vector for the not selected field picture coded in one pel units according to Table 7-4g in Section 7. This element is present only if the **field\_or\_frame\_backward** flag is set to 1.

**dmv\_vertical\_backward** -- Difference between a vertical motion vector reconstructed from a backward vertical motion vector for the selected field picture and a backward

vertical motion vector for the not selected field picture coded in one pel units according to Table 7-4g in Section 7. This element is present only if the field\_or\_frame\_backward flag is set to 1.

**coded\_block\_pattern** -- The variable cbp is derived from the coded\_block\_pattern using the variable length code Table 7-3 in Section 7. Then, the pattern\_code[i] for i=0 to 3 is derived from cbp using the following:

```
pattern_code[i] = 0;
if ( cbp & (1<<(3-i)) ) pattern_code[i] = 1;
if ( macroblock_intra ) pattern_code[i] = 1;
```

pattern\_code[0] -- If 1, then the left luminance block is to be received in this macroblock.

pattern\_code[1] -- If 1, then the right luminance block is to be received in this macroblock.

pattern\_code[2] -- If 1, then the chrominance difference block Cb is to be received in this macroblock.

pattern\_code[3] -- If 1, then the chrominance difference block Cr is to be received in this macroblock.

The meaning of the following data elements is the same as in MPEG1 C.D. on 5/31/91.

macroblock_stuffing	macroblock_escape
macroblock_address_increment	macroblock_type
end_of_macroblock	

The meaning of the following data elements is the same as in MPEG1 C.D. on 12/18/90.

motion_horizontal_forward	motion_vertical_forward
motion_horizontal_backward	motion_vertical_backward

### 3.2.7 Block layer

**dct\_coeff\_first** -- A variable length code according to Tables 7-5c through 7-5g in Section 7 for the first coefficient. The zigzag-scanned quantized dct coefficient list is updated as follows.

```
i = run;
if ( s == 0 )  dct_zz[i] = level;
if ( s == 1 )  dct_zz[i] = -level;
```

The terms dct\_coeff\_first and dct\_coeff\_next are the run-length encoded and dct\_zz[i], i>0 shall be set to zero initially. A variable length code according to Tables 7-5c through 7-5g and 7-5m is used to represent the run and level of the DCT coefficients.

**dct\_coeff\_next** -- A variable length code according to Tables 7-5h through 7-5m in Section 7 for coefficients following the first retrieved. The zig-zag scanned quantized dct coefficient list is updated as follows.

```

i = i + run + 1 ;
if ( s == 0 )    dct_zz[i] = level ;
if ( s == 1 )    dct_zz[i] = -level;

```

If **macroblock\_intra** == 1, then the term **i** shall be set to zero before the first **dct\_coeff\_next** of the block.

The meaning of the following data elements is the same as in MPEG1 C.D. on 5/31/91.

<b>dct_dc_size_luminance</b>	<b>dct_dc_size_chrominance</b>
<b>dct_dc_differential</b>	<b>end_of_block</b>

### 3.3. The decoding process

#### 3.3.1 Intra-coded macroblocks

All blocks are intra-coded and transmitted in the I-pictures . Some macroblocks may be intra-coded as identified by `macroblock_type` in the P0-pictures, P1-pictures, P2-pictures and B-pictures . Thus, `macroblock_intra` identifies the intra-coded macroblocks.

The discussion of semantics has defined `mb_row` and `mb_column`, which locate the macroblock in the picture. The definitions of `dct_dc_differential` and `dct_coeff_next` have also defined the zigzag-scanned quantized dct coefficient list, `dct_zz[]`. Each `dct_zz[]` is located in the macroblock as defined by `pattern_code[]`.

Let us define `dct_recon[8][8]` to be the matrix of the reconstructed dct coefficients, where the first index identifies the row and the second the column of the matrix. Define `dct_dc_y_past`, `dct_dc_cb_past`, and `dct_dc_cr_past` to be the `dct_recon[0][0]` of the most recently decoded intra-coded Y, Cb, and Cr blocks respectively. The predictors `dct_dc_y_past`, `dct_dc_cb_past`, and `dct_dc_cr_past` are all reset at the start of a slice and at non-intra-coded macroblocks (including skipped macroblocks) to the value 128.

Define `intra_quant[8][8]` to be the intra quantizer matrix specified in the sequence header. Note - `intra_quant[0][0]` is used in the inverse quantizer calculations for simplicity of description, but the result is overwritten by the subsequent calculation for the dc coefficient.

Define `non_intra_quant[8][8]` to be the non-intra quantizer matrix specified in the sequence header.

Define `scan[4][8][8]` to be the matrix defining the zigzag scanning sequence.

The first 8x8 elements of the matrix ( `scan[0][][ ]` ) are used for the intra-coded luminance block and are defined as follows :

0	2	6	12	20	28	34	50
1	4	11	19	27	33	35	51
3	8	17	25	31	36	49	52
5	10	18	26	32	37	48	53
7	15	23	30	39	38	47	54
9	16	24	40	45	46	55	60
13	21	29	41	44	56	59	61
14	22	42	43	57	58	62	63

The second 8x8 elements of the matrix ( `scan[1][][ ]` ) are used for the intra-coded chrominance block and are defined as follows :

0	1	5	6	14	15	27	28
2	4	7	13	16	26	29	42
3	8	12	17	25	30	41	43
9	11	18	24	31	40	44	53

10	19	23	32	39	45	52	54
20	22	33	38	46	51	55	60
21	34	37	47	50	56	59	61
35	36	48	49	57	58	62	63

The third 8x8 elements of the matrix ( scan[2][[]] ) are used for the non-intra-coded luminance block and are defined as follows :

0	8	16	24	32	40	48	56
1	9	17	25	33	41	49	57
2	10	18	26	34	42	50	58
3	11	19	27	35	43	51	59
4	12	20	28	36	44	52	60
5	13	21	29	37	45	53	61
6	14	22	30	38	46	54	62
7	15	23	31	39	47	55	63

The fourth 8x8 elements of the matrix ( scan[3][[]] ) are used for the non-intra-coded chrominance block and are defined as follows :

0	1	2	3	4	7	15	34
5	6	8	10	14	26	35	56
9	11	13	16	25	36	43	58
12	18	17	24	30	42	50	59
19	23	27	29	41	48	52	60
20	28	31	37	44	49	54	61
21	32	38	40	46	51	55	62
22	33	39	45	47	53	57	63

Define past\_intra\_address as the macroblock\_address of the most recently retrieved intra-coded macroblock within the slice. It is reset to -2 at the beginning of each slice.

Then dct\_recon[8][8] is computed by any means equivalent to the following procedure for the first luminance block:

```

for (m=0; m<8; m++) {
    for (n=0; n<8; n++) {
        i = scan[0][m][n];
        dct_recon[m][n] = ( 2 * dct_zz[i] * ( 64 + quantizer_scale
                                           * intra_quant[m][n] ) ) / 16;
        if ( ( dct_recon[m][n] & 1 ) == 0 )
            dct_recon[m][n] = dct_recon[m][n] - Sign(dct_recon[m][n])
;
        if (dct_recon[m][n] > 2047) dct_recon[m][n] = 2047;
        if (dct_recon[m][n] < -2048) dct_recon[m][n] = -2048;
    }
}
dct_recon[0][0] = dct_zz[0] * 8;
if ( ( macroblock_address - past_intra_address > 1 ) )
    dct_recon[0][0] = 128 * 8 + dct_recon[0][0];

```

```

else
    dct_recon[0][0] = dct_dc_y_past + dct_recon[0][0];
    dct_dc_y_past = dct_recon[0][0];

```

For the second luminance block in the macroblock, in the order of the pattern\_code list:

```

for (m=0; m<8; m++) {
    for (n=0; n<8; n++) {
        i = scan[0][m][n];
        dct_recon[m][n] = ( 2 * dct_zz[i] * ( 64 + quantizer_scale
                                     * intra_quant[m][n] ) ) / 16;
        if ( ( dct_recon[m][n] & 1 ) == 0 )
            dct_recon[m][n] = dct_recon[m][n] - Sign(dct_recon[m][n])
;
        if (dct_recon[m][n] > 2047) dct_recon[m][n] = 2047;
        if (dct_recon[m][n] < -2048) dct_recon[m][n] = -2048;
    }
    dct_recon[0][0] = dct_dc_y_past + dct_zz[0] * 8;
    dct_dc_y_past = dct_recon[0][0];

```

For the chrominance Cb block,:

```

for (m=0; m<8; m++) {
    for (n=0; n<8; n++) {
        i = scan[1][m][n];
        dct_recon[m][n] = ( 2 * dct_zz[i] * ( 64 + quantizer_scale
                                     * intra_quant[m][n] ) ) / 16;
        if ( ( dct_recon[m][n] & 1 ) == 0 )
            dct_recon[m][n] = dct_recon[m][n] - Sign(dct_recon[m][n]);
        if (dct_recon[m][n] > 2047) dct_recon[m][n] = 2047;
        if (dct_recon[m][n] < -2048) dct_recon[m][n] = -2048;
    }
    dct_recon[0][0] = dct_zz[0] * 8;
    if ( ( macroblock_address - past_intra_address > 1 ) )
        dct_recon[0][0] = 128 * 8 + dct_recon[0][0];
    else
        dct_recon[0][0] = dct_dc_cb_past + dct_recon[0][0];
    dct_dc_cb_past = dct_recon[0][0];

```

For the chrominance Cr block, :

```

for (m=0; m<8; m++) {
    for (n=0; n<8; n++) {
        i = scan[1][m][n];
        dct_recon[m][n] = ( 2 * dct_zz[i] * ( 64 + quantizer_scale
                                     * intra_quant[m][n] ) ) / 16;
        if ( ( dct_recon[m][n] & 1 ) == 0 )
            dct_recon[m][n] = dct_recon[m][n] - Sign(dct_recon[m][n]);
        if (dct_recon[m][n] > 2047) dct_recon[m][n] = 2047;

```

```

        if (dct_recon[m][n] < -2048) dct_recon[m][n] = -2048 ;
    }
}
dct_recon[0][0] = dct_zz[0] * 8 ;
if ( ( macroblock_address - past_intra_address > 1 ) )
    dct_recon[0][0] = 128 * 8 + dct_recon[0][0] ;
else
    dct_recon[0][0] = dct_dc_cr_past + dct_recon[0][0] ;
dct_dc_cr_past = dct_recon[0][0] ;

```

After all the blocks in the macroblock are processed:

```
past_intra_address = macroblock_address ;
```

Once the dct coefficients are reconstructed, the inverse DCT transform is applied to obtain the inverse transformed pel values in the range [-256, 255]. These pel values must be clipped to the range [0, 255] and are placed in the luminance and chrominance matrices in the positions defined by mb\_row, mb\_column, and the pattern\_code list.

### 3.3.2 Predictive-coded macroblocks in P0- and P1-pictures

Predictive-coded macroblocks in P0- and P1-pictures are decoded in two steps. First, the value of the forward motion vector for the macroblock is reconstructed and a prediction macroblock is formed, as detailed below. Second, the DCT coefficient information stored for some or all of the blocks is decoded, inverse DCT transformed, and added to the prediction macroblock.

Let recon\_right\_for[] and recon\_down\_for[] be the reconstructed horizontal and vertical components of the motion vector for the current macroblock, and recon\_right\_for\_prev[] and recon\_down\_for\_prev[] be the reconstructed motion vector for the previous predictive-coded macroblock. The index of the matrix may be either 0 or 1. If the index equals 0, the motion vectors are used for reconstructing the even field reference picture. If the index equals 1, the motion vectors are used for reconstructing an odd field reference picture. If this is the first macroblock in the slice, or if the last macroblock contained no motion vector information (either because it was skipped or macroblock\_motion\_forward was zero), then recon\_right\_for\_prev[] and recon\_down\_for\_prev[] shall be set to zero.

If no forward motion vector data exists for the current macroblock (either because it was skipped or macroblock\_motion\_forward == 0), the motion vectors shall be set to zero.

If forward motion vector data exist for the current macroblock, the following procedure is used to reconstruct the motion vector horizontal and vertical components. Decoded values right\_little, right\_big, down\_little and down\_big are found from an appropriate Table 7-4a through 7-4f in Section 7.

Let `fi_or_fr_for_prev` be the flag of the interpolation mode a the previous-coded macroblock. If this is the first macroblock in the slice, or if the last macroblock decoded contained no motion vector information, then `fi_or_fr_for_prev` shall be set to zero.

Let `sel_mv_for_prev` be the flag of `select_mv_forward` for a previous predictive-coded macroblock. If this is the first macroblock in the slice, or if the last macroblock decoded contained no motion vector information, then `sel_mv_for_prev` shall be set to zero.

Let `dist[0]` be the distance between an even field reference picture and a current decoding picture. Let `dist[1]` be the distance between an odd field reference picture and a current decoding picture.

Then the motion vector in half-pel units is reconstructed as follows:

```

max = ( 16 * forward_f ) - 1 ;
min = ( -16 * forward_f ) ;

if ( fi_or_fr_for_prev == 0 ) {
    new_vector = recon_right_for_prev[sel_mv_for_prev] + right_little ;
    if ( new_vector <= max && new_vector >= min )
        recon_right_for[select_mv_forward]
            = recon_right_for_prev[sel_mv_for_prev] + right_little ;
    else
        recon_right_for[select_mv_forward]
            = recon_right_for_prev[sel_mv_for_prev] + right_big ;
    recon_right_for_prev[select_mv_forward]
        = recon_right_for[select_mv_forward] ;

    new_vector = recon_down_for_prev[sel_mv_for_prev] + down_little ;
    if ( new_vector <= max && new_vector >= min )
        recon_down_for[select_mv_forward]
            = recon_down_for_prev[sel_mv_for_prev] + down_little ;
    else
        recon_down_for[select_mv_forward]
            = recon_down_for_prev[sel_mv_for_prev] + down_big ;
    recon_down_for_prev[select_mv_forward]
        = recon_down_for[select_mv_forward] ;
}
else {
    new_vector = recon_right_for_prev[select_mv_forward] + right_little ;
    if ( new_vector <= max && new_vector >= min )
        recon_right_for[select_mv_forward]
            = recon_right_for_prev[select_mv_forward] + right_little ;
    else
        recon_right_for[select_mv_forward]
            = recon_right_for_prev[select_mv_forward] + right_big ;
    recon_right_for_prev[select_mv_forward]
        = recon_right_for[select_mv_forward] ;

    new_vector = recon_down_for_prev[select_mv_forward] + down_little ;

```

```

        if ( new_vector <= max && new_vector >= min )
            recon_down_for[select_mv_forward]
                = recon_down_for_prev[select_mv_forward] + down_little
        ;
        else
            recon_down_for[select_mv_forward]
                = recon_down_for_prev[select_mv_forward] + down_big ;
            recon_down_for_prev[select_mv_forward]
                = recon_down_for[select_mv_forward] ;
    }

    if ( field_or_frame_forward == 1 ) {
        recon_right_for[!select_mv_forward]
            = 2 * ( ( (double) recon_right_for[select_mv_forward] / 2.0 *
                    dist[!select_mv_forward] / dist[select_mv_forward] ) / 1
                - dmv_horizontal_forward ) ;
        recon_down_for[!select_mv_forward]
            = 2 * ( ( (double) recon_down_for[select_mv_forward] / 2.0 *
                    dist[!select_mv_forward] / dist[select_mv_forward] ) / 1
                - dmv_vertical_forward ) ;
        recon_right_for_prev[!select_mv_forward]
            = recon_right_for[!select_mv_forward] ;
        recon_down_for_prev[!select_mv_forward]
            = recon_down_for[!select_mv_forward] ;
    }

    if ( select_mv_forward == 0 && ( dist[0] % 2 == 1 ) )
        recon_down_for[select_mv_forward] ++ ;
    if ( select_mv_forward == 1 && ( dist[0] % 2 == 0 ) )
        recon_down_for[select_mv_forward] -- ;
    fi_or_fr_for_prev = field_or_frame_forward ;
    sel_mv_for_prev = select_mv_forward ;

```

The motion vectors in integer pel units for the macroblock, right\_for[] and down\_for[], and the half pel unit flags, right\_half\_for[] and down\_half\_for[], are computed as follows:

for luminance	for chrominance
right_for[] = recon_right_for[] >> 1 ;	right_for[] = ( recon_right_for[] / 2 ) >> 1 ;
down_for[] = recon_down_for[] >> 1 ;	down_for[] = ( recon_down_for[] / 2 ) >> 1 ;
right_half_for[] = recon_right_for[] - 2*right_for[] ;	right_half_for[] = recon_right_for[]/2 - 2*right_for[] ;
down_half_for[] = recon_down_for[] - 2*down_for[] ;	down_half_for[] = recon_down_for[]/2 - 2*down_for[] ;